Exhibit 1

Greenhouse UI Architecture

1. Overall

Almost all of the client side pages will rely on an ALC JavaScript library to standardize operations like positioning, dynamic content loading, and dynamic content generation on the client side. This will allow us to do this in a cross browser way if we decide to. See TBD TBD for the JavaScript Library requirements.

We also need a single point of entry that handles login and establishes session information.

Need to develop:

1) Login Screen
2) Login Servlet
3) JavaScript Library that includes:
   a) Creation of screen elements from JS Objects (both at load time and while running)
   b) Dynamic update of screen element content.
   c) Cross browser screen element style access.
   d) JS Behaviors (motion, click handlers, onLoad handler chains)
   e) Naming and Locating (both screen elements and JSObjects)

2. Navigation Pane

The Navigation Pane contains three active areas: the Show Menu, the Tree, and the Navigation Tabs. The operation of this pane is very consistent for all four tabs (almost identical for Geographic and Network). This should be one page where the tree control is placed into different states (views) by the selection of navigation tabs. The selection of navigation tabs should not cause the left pane to reload. This makes it easier for the tree to keep state information when switching tabs, then back again. The tree should only get as much information as it needs from the server to display the current level (perhaps some optimization later to get the next one?). The tree should retain information from a collapsed branch that was previously expanded. It should also be able to support some type of reset function. The Tree control will probably be a Java Applet (it can be done in JavaScript, but probably isn't worth it). *We should do some more testing on LiveConnect before deciding this finally. Try LiveConnect on several IE platforms and on NS under Linux.* Should the tree poll the core for updates to already downloaded information? We won't support real time editing of the tree in version 1.0, but at least the icons can change. Perhaps this refresh could be triggered "manually" when a tree altering operation is performed. We will also need a servlet to look up the tree information on the server. The servlet should respond to queries with tree content information formatted in XML. When a selection is made in the tree, the tree control should navigate the action pane to the appropriate URL.

The XML tree information needs to contain a hierarchical tree of nodes where each node has:

1) Unique ID
2) Leaves marked.
3) Icon URL for collapsed and expanded state (do these need to be different for some nodes?)
4) List of associated icons w/ icon GUIDs and icon URLs
5) Node navigation URL
6) Node Text

The tree control applet must keep a set of tree information for each of the four tabs.

In summary, the navigation pane consists of a single normal HTML page populated with three major elements.

Need to develop:

1) A good tree control applet.
2) A JavaScript Drop down menu that can send control information to the tree.
3) A JavaScript Tabset that can send control information to the tree.
4) A Single HTML page containing the above elements which dynamically sizes the tree to fill the appropriate space.
5) A TreeInfo servlet.

3. Action Pane

For the action pane, the content of each tab is very different. Selection of different tabs on this page should navigate the action pane to a new page. The currently selected tab should also be stored (in a JavaScript object in the nav pane) so that the selected tab may stay selected when a new node is selected. The action pane will be a single frame (initially targeted to a servlet). The tabset at the top and any border will be generated by the action pane servlet – the remainder of the page should be obtained using JSP chaining. The action pane servlet should take as a parameter, the id of the selected node and store it in the session object. The tabset will always include the same number of tabs – inactive tabs for a particular node will be grayed out.

Need to develop:

1) An ActionPane servlet which can dynamically generate the appropriate tabset.
2) Architecture to chain JSP (or another servlet) from the servlet to get the contents.

3.1. Graphics

The target URL for the graphics content should actually be another servlet, not a JSP page. This servlet should find the appropriate page for the node. If a custom graphics page is indicated in the database, it will simply chain to it, otherwise it will chain to the appropriate object control JSP or child list JSP. All of the graphics pages will contain dynamic content which will be updated separately from the main page via a hidden

(inline) frame that just evaluates JavaScript. This frame will be pointed at an expression evaluation servlet that returns JavaScript. The expressions to be should be read from a server side XML conditions file associated with that graphics page. The selected node's ID should be read out of the session object.

Need to develop:

1) Expression evaluation servlet
2) Graphics page template
3) Dynamic Color Server (for thermographic floorplans)
4) Collection of Dynamic elements to place on page (JavaScript and Fusion components)
   a) Text (Content, Style, Color)
   b) Graphics (Dynamic Source)
   c) Animation (Start / Stop)
   d) Dynamic Color Floorplans

Need to develop JSPs and controls for the following default object pages:
(each of these may have different flavors for different units)

1) Analog Input
2) Analog Output
3) Analog Value
4) Binary Input
5) Binary Output
6) Binary Value
7) Multi-State Input
8) Multi-State Output
9) Multi-State Value
10) Calendar
11) Schedule

3.2. Properties

This process needs some work! We need to expriment with the property value access from within the JSP.

The property page is a JSP page generated by Eikon at design time for each function block. The page should utilize JSPs JavaBean access to get the information for all of the variable content. All of the data access for the page should be optimized into one web-server to core CORBA call. Properties are displayed using JavaScript library functions that allow a non-form look and feel.

Customization idea – start with pre-made JSPs for all microblocks. Each JSP accepts a few standard parameters such as "name". Basic Eikon customization is just to set design time parameters which are then generated as a part of the resulting JSP files for each GFB. For a more customized microblock property page, the system should copy (and uniquely name) the default microblock JSP page and start the HTML editor

(Dreamweaver?) with the new file. We should develop some DW extensions to allow easy access to the properties.

Need to develop:

1) JavaBeans to look up values (including lookup synchronization)
2) JavaScript library of property components.
3) Property JSPs for all microblocks.

3.3. Schedule

The scheduling tab will largely be an application written in JavaScript. All of the user interaction code will run in JavaScript on the client. However, any code for database or core access should be done in a servlet or a server side JavaBean. The scheduling tab will point to a frameset page. The top frame will contain a JSP with data generated on the server. This frame will reload to get new data. The bottom right frame (calendar) will probably be a Java Applet. *We should look for some good existing calendar applets (commercial or public domain).* The editing / results page will be another JSP page. Schedule evaluation will be done in the core and accessed through a server side JavaBean. Editing / display of the time lines will be done with either JavaScript or several Java Applets.

Need to develop:

1) JavaBean to build list of applicable schedules for a level.
2) Schedule Frameset
3) Top JSP page w/ JavaScript
4) Calendar Control
5) Timeline Edit Control
6) Edit / Display JSP page w/ JavaScript Logic

3.4. Events

3.5. Reporting Actions

3.6. Trends

The trending tab will consist of two JSP pages, one for configuration and one for viewing. The images on the viewing tab will come from a TrendView servlet, which may be referenced in other locations as well. The method of editing the trend view parameters has not been determined.

Need to develop:

1) Two Trend JSPs
2) Trend View Servlet